REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

AD-A263 144

2. REPORT DATE	March 1993
3. REPORT TYPE AND DATES COVERED	
4. TITLE AND SUBTITLE	Understanding the Applicability of Sequential Data Analysis Techniques for Analyzing Usability Data
5. FUNDING NUMBERS	
6. AUTHOR(S)	Donna L. Cuomo
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	The MITRE Corporation 202 Burlington Road Bedford, MA 01730-1420
8. PERFORMING ORGANIZATION REPORT NUMBER	M 93B0000018
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	The MITRE Corporation 202 Burlington Road Bedford, MA 01730-1420
10. SPONSORING/MONITORING AGENCY REPORT NUMBER	M 93B0000018
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT	Approved for public release; distribution unlimited
12b. DISTRIBUTION CODE	A

13. ABSTRACT *(Maximum 200 words)*

The applicability of exploratory sequential data analysis (ESDA) techniques for analyzing usability test data is examined. ESDA techniques include transition matrix analysis, lag sequential analysis, frequency of cycles, graphical summarization techniques, and pattern analysis techniques. A subset of each was used in analyzing the data from three usability studies. The encoding schemes used, the analysis routines run, software tools to support encoding and analysis (SHAPA and the Maximal Repeating Pattern analysis tool), and their interactions are discussed and the different types of usability problems which can be extracted from the data when analyzed with ESDA techniques are illustrated. It is concluded that the ESDA techniques will be useful once the state of the art in software support is able to provide the analyst greater flexibility in applying the analysis routines.

14. SUBJECT TERMS	ESDA; sequential data analysis; SHAPA; Maximal Repeating Pattern analysis tool
15. NUMBER OF PAGES	15
16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT	Unclassified
18. SECURITY CLASSIFICATION OF THIS PAGE	Unclassified
19. SECURITY CLASSIFICATION OF ABSTRACT	Unclassified
20. LIMITATION OF ABSTRACT	

Understanding the Applicability of Sequential Data Analysis Techniques for Analyzing Usability Data

M 93B0000018
March 1993

Donna L. Cuomo

20p8

MITRE

202 Burlington Road
Bedford, Massachusetts 01730-1420

Understanding the Applicability of Sequential Data Analysis Techniques for Analyzing Usability Data

M 93B0000018
March 1993

Donna L. Cuomo

Contract Sponsor MSR
Contract No. N/A
Project No. 9162A
Dept. D047

MITRE

202 Burlington Road
Bedford, Massachusetts 01730-1420

Department Approval: [illegible]
Nancy C. Goodwin

MITRE Project Approval: [illegible]
Donna L. Cuomo

[illegible]

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Table of Contents

List of Figures

List of Tables

Abstract

The applicability of exploratory sequential data analysis (ESDA) techniques for analyzing usability test data is examined. ESDA techniques include transition matrix analysis, lag sequential analysis, frequency of cycles, graphical summarization techniques, and pattern analysis techniques. A subset of each was used in analyzing the data from three usability studies. The encoding schemes used, the analysis routines run, software tools to support encoding and analysis (SHAPA and the Maximal Repeating Pattern analysis tool), and their interactions are discussed and the different types of usability problems which can be extracted from the data when analyzed with ESDA techniques are illustrated. It is concluded that the ESDA techniques will be useful once the state of the art in software support is able to provide the analyst greater flexibility in applying the analysis routines. Without the ability to apply analysis routines to multiple data levels, too much work is involved in obtaining a complete analysis of usability problems at all levels.

1 Introduction

A potential problem with some usability test methods, such as verbal protocol techniques and user observation, is subjectivity in the data analysis and interpretation (Holleran 1991: 352). To address this problem, we have been experimenting over the years with understanding the applicability of exploratory sequential data analysis (ESDA) techniques for analyzing usability data. ESDA is a family of tools and techniques for exploring sequential data collected in complex, dynamic, event-driven environments (Sanderson 1991). Applying the techniques typically involves transcribing and encoding recorded events, and applying statistical analysis routines such as Markov analysis, lag sequential analysis, cycle analysis, and pattern analysis techniques to the encoded data. It has been proposed by various researchers (Siochi et al. 1991, Holleran 1991) that such techniques could prove useful in the area of human-computer interaction analysis.

While several studies have been documented in which Markov analysis, for example, has been used (e.g., Hammer and Rouse 1979, Penniman 1975, and Good 1985), we were unable to find a comprehensive guide or discourse on the various ESDA techniques available and how they should be used in the context of usability testing. If these types of techniques prove useful in usability test data analysis, they would enhance the process of converting logged usability test data into information that is less subjective, and more rigorous and quantifiable, and would permit the use of automated analysis tools. This paper describes three systems for which we have performed usability tests and applied a subset of these ESDA techniques during the data analysis phase, and our lessons learned from these experiences. Our intent is to provide a perspective in this area that will help other usability analysts decide whether it is worth the effort to apply these techniques, and which techniques are most likely to produce meaningful results for them. Our experiences may also help developers of usability test tools understand the practitioner's needs, in terms of the type of information we should extract from our usability data when evaluating the usability of software systems.

2 ESDA Techniques

Sanderson (1991) explains ESDA as encompassing concepts from exploratory data analysis philosophy, sequential data analysis (SDA) techniques, and human-computer interaction. She proposes ESDA as a way of analyzing data that is rich, complex and multi-dimensional and cannot be readily analyzed with conventional statistical techniques. Characteristics of such data are that events unfold over time and preservation of the temporal dimension is important, the data can usually be analyzed at many different levels, and you may not initially know the questions to be answered. SDA techniques include methods for sampling, coding and analysis. The analytic methods include time series, Markov, lag sequential, causal, cycle, grammars (Sanderson et al. 1991) and pattern analysis and identification techniques. Below we briefly describe the techniques we have used, whether or not data needs to be encoded and the manual versus software-aided options for data encoding and applying SDA techniques.

2.1 Sequential Data Analysis

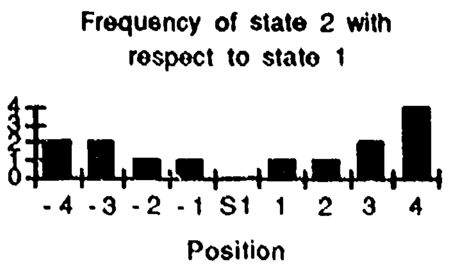
The movement between states can be explored by modeling them as a finite Markov chain, which is defined by Kemeny and Snell (1960: 201) as "a stochastic process which moves through a finite number of states, and for which the probability of entering a certain state depends only on the last state occupied." Matrix analyses involve constructing transition frequency or probability matrices to examine whether there are dependencies in the data. The analysis may reveal habitual or stereotyped patterns of behavior (Sanderson et al. 1989).

Lag sequential analyses (originally developed by Sackett 1974, in Sanderson et al. 1989) permit vague patterns to be discerned in a data sequence. Such analyses display the frequency with which one state occurs with respect to another state, at various removes. If state 1 is the target state, the analysis displays the frequency with which state 2 occurs directly before state 1, two steps before state 1, directly after state one, two steps after state 1, etc. This helps determine patterns of behavior that may not be strictly sequential (Sanderson et al. 1989).

Frequency of cycles (originally developed by Fisher 1988, in Sanderson et al. 1989), looks for regularities in behavior sequences. This form of analysis provides a report of actually occurring sequences of commands or states in a single cycle defined by a target command or state. That is, if the target state is state 1, the first and second occurrences of state 1 are identified and all the events in between are stored as a cycle. This is repeated for the second and third occurrences of state 1, and so on until all the cycles are identified. Each cycle is then compared for matches, and the frequencies of each cycle are counted. The result is a listing of all cycles, the number of times each occurred, and the sequence in which they occurred (Sanderson et al., 1989). Each of these is illustrated in figure 1.

	E	D	S	P	Dr	S	D	B
Error	.15	.05	.00	.05	.18	.54	.03	.00
Draft	.19	.15	.00	.08	.08	.50	.00	.00
Sp/Join	.5	.00	.00	.00	.00	.00	.00	.50
Percep	.00	.57	.00	.05	.00	.38	.00	.00
Draw	.02	.00	.00	.00	.68	.19	.10	.00
System	.06	.03	.01	.06	.32	.46	.01	.06
Delete	.08	.02	.00	.00	.49	.16	.24	.02
Back	.06	.00	.00	.13	.13	.69	.00	.00

(a)



(b)

#	Freq	Cycle
A.	1	Int.task -> eval. -> int.task
B.	1	Int.task-> int.exec -> eval. -> int.percept -> eval. -> eval. -> int.task
C.	5	Int.task -> int.exec -> tape ->tape -> command -> button -> eval. -> int.exec -> menu -> command -> eval. -> eval. -> int.task

(c)

Figure 1. Outputs from Three SDA Techniques,
(a) First-Order Transition Matrix, (b) Lag Sequential Analysis, and (c) Frequency of Cycles.

Another pattern analysis technique, maximal repeating patterns (MRP), was developed by Siochi and Ehrich (1991). MRP analysis works on the premise that repetition of user actions is an important indicator of potential user interface problems. The technique detects the longest repeating sequences of command strings in user session transcripts; it works on the user inputs only. The method is unique in that the analyst does not identify a priori the patterns or strings the technique should search for; all repeating patterns are identified.

2.2 *Graphical Summarization*

We have previously proposed graphical summarization techniques to aid in analyzing sequential data in the computer-aided design task domain (Cuomo and Sharit 1989). These ideas, based on the human problem solving concepts presented by Newell and Simon (1977), included task movement graphs, task rate graphs, and analysis of inter-event intervals, together with Markov process analysis. The graphical output of these routines is illustrated in figure 2.

The first two graphical techniques are based on the concept of moving through a design or task state space toward a goal. Recorded subject-input events are coded in terms of whether the event moved the user a step forward toward a goal, whether the command was a result of using a computer and did not move the user forward a step (e.g., moving a window), or whether the event caused the user to move backwards, away from the end goal (e.g., delete something). The overall slope of the graph indicates the efficiency with which a task is performed. Rate measures can also be constructed which show absolute forward movement of task progress over a time period, where time periods are the total time for discrete task segments. Finally, inter-event interval analysis involved plotting the elapsed time between subject-input events against interval numbers. Long delays between events can then be identified, flagging potential problem areas requiring further investigation.

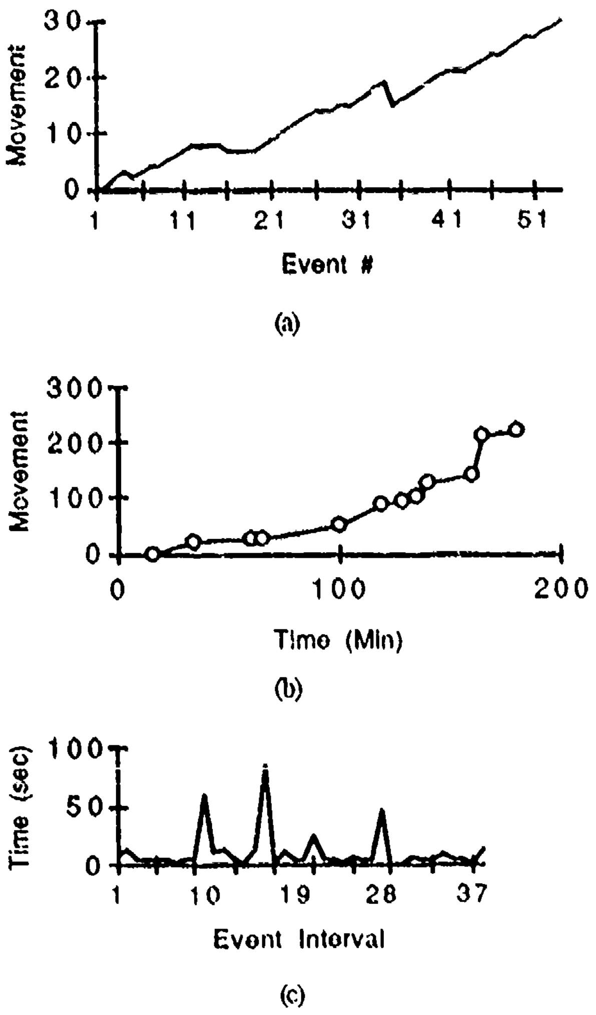


Figure 2. Outputs From Three Graphical Summarization Techniques, (a) Task Movement Graph, (b) Task Rate Graph, and (c) Inter-Event Interval Graph.

To apply any of these analytic routines, software can be written in any conventional software language and results output and displayed with commercial database and graphing packages. For some routines, e.g., inter-event interval analysis, this approach is straightforward and easy. Having several stand-alone analysis routines, however, does not readily support the exploratory aspects required for a complete analysis of the data, and writing analysis code can add time to an already time-consuming process.

Nevertheless there may not be alternatives. Few mature software tools designed to support ESDA techniques are available. The tools we were able to obtain are discussed later in the paper.

2.3 Encoding the Data

The type of data collected, the analytic technique itself, and the questions to be answered determine whether the technique can be applied directly to the collected usability data, or whether the data needs to be encoded first. When collecting verbal protocols, for instance, the content of the sentences and actions of interest need to be extracted and encoded before ESDA techniques can be applied; statistical techniques obviously cannot be applied directly. When files of actual mouse and keystrokes or command input files are collected, some of the analysis techniques can be applied without encoding the data. Siochi et al. (1991), for instance, applied their MRP routines to unencoded command input files; such files may need some filtering to be put in a form acceptable to the analysis program. The task rate and task movement graphs, although applied to similar types of user input files, must be encoded as the task movement implications of each input must be determined. For a technique such as transition matrices, the purpose of the analysis will determine whether the data needs to be encoded. If the goal is to determine the most commonly occurring input pairs, for instance, the technique should be applied to the actual user input sequences. If patterns of higher-level behavior are sought, the data should be coded appropriately; in fact data from several sources may be used to support application of the encodings to the data.

When encoding data files, selection of the encoding scheme is a critical factor influencing the questions that can be answered by the data. An appropriate scheme is usually established iteratively. Once decided upon, the analyst can apply it manually by examining the data files, choosing the appropriate code for each input or group of inputs, and typing the software codes into a new file. Alternatively, tools to support data encoding can be written, or existing tools used.

2.4 Software for Supporting Data Encoding and SDA Techniques

As it is not practical to apply these ESDA techniques or complex encoding schemes manually, software support is required for both the data encoding and application of the analysis routines. How the software applies the techniques to the data, and the amount of flexibility in the software, however, affect the resultant usefulness of the analysis routine outputs.

One tool we were able to obtain that supports both encoding and analysis was SHAPA (Software for Heuristically Aiding Protocol Analysis) Version 2.0 , developed by the University of Illinois at Urbana-Champaign Engineering Psychology Research Laboratory. SHAPA is a protocol analysis environment permitting researchers to encode data in any way they choose. It works on single-stream, un-timestamped verbal and non-verbal protocols, running on an IBM PC or compatible. A new version currently under development, MACSHAPA, will include the ability to analyze multiple stream, time-stamped data from video records as well as ASCII files (Sanderson et al. 1991).

To encode data with SHAPA, the researcher first decides on the encoding scheme, then specifies the codes, called predicates, on the predicates screen. A predicate consists of the name followed by its arguments or values in parentheses, and might look like: INT.TASK (1-1.1-setting), or MENU(view, m). By having the data analyst predefine the predicates and their values, SHAPA can allow the user to type in an abbreviation for each predicate, leaving the software to complete the name. Once all the files are encoded, running the analysis routines is as easy as selecting the desired routine from the Reports menu. SHAPA supports transition matrix analysis, lag sequential analysis, frequency of cycles, value lists, and predicate instances. The value list routine generates a report which, for each predicate, lists all the constant values used as predicate arguments and their frequency of use. The collection of predicate instances routine collects segments encoded with the same predicate and displays the line number on which they occurred; values can also be specified as being of interest (James et al. 1990). Except for these last two routines, which do take account of the predicate values, the other routines all report on predicates only, not their specific values. (See any of the Sanderson et al. references or James, Sanderson, and Seidler 1990 for more detailed descriptions of SHAPA and its capabilities.)

A second software package we were able to obtain for our usability lab was the Maximal Repeating Pattern (MRP) Tool developed at the Department of Computer Science at Virginia Polytechnic Institute and State University, which supports extraction of MRPs from logged user input data. We used the version designed to run on UNIX-based systems. To use the application, the logged user input data is normalized to convert raw transcripts to a standard form. Siochi et al. (1991) used the tool to analyze data patterns on collected user sessions on a command-based image processing system called GIPSY. They normalized the raw transcript files by extracting inputs from the collected user inputs and system outputs, and then extracted single-word commands from the command argument pairs in the transcript. The software then identifies all the MRPs in the file. MRPs are defined as repeating patterns that are as long as possible, or are independently occurring substrings of longer patterns (Siochi et al. 1991: 316). Their outputs include each identified MRP in order of decreasing length, and the number of instances. Summary information includes the number of MRPs found and their minimum, maximum, and average length. The analyst can filter MRPs, examine specific instances of each, and get more details on a specific one, following a pointer back to its instance in the raw transcript file. This technique tends to generate large amounts of data which need to be filtered. The MRP developers estimate that one MRP is found for every 20 to 25 command lines.

3 Usability Testing of Three Software Systems

In the past four years, we have conducted three formal usability studies on a variety of systems and, for each, have used some subset of ESDA techniques to analyze the collected usability data. All systems were similar in possessing graphical, direct manipulation style user interfaces. All supported realistic tasks that were user-controlled as opposed to system-controlled, and none of the tasks were time critical. All the tasks were also ill-defined, in the sense that there could be many correct solutions to the problems; we expected high degrees of variability across participants in terms of both the problem-solving approach taken and the computer-use strategies. Task times for each test ranged from one and a half hours to approximately four hours. Finally, all participants were representative of the intended user population, and the participants in each test were trained to use the system by the usability testers. They had not used the systems previously.

Each system and the usability test procedures used are described briefly below. Detailed descriptions of procedure and usability problems identified can be found in the referenced usability test reports. Our focus here is on the way in which data encoding was performed and the effectiveness of applying ESDA techniques.

3.1 Computer-Aided Architectural Design System

A usability study of a commercial computer-aided architectural design (CAAD) system was performed, in which six architectural design students performed two design tasks at one of three levels of complexity (Cuomo and Sharit 1989). Although verbal protocols, user keystrokes and stylus inputs were collected, only the keystrokes and stylus inputs were used for the application of SDA techniques.

Two encoding schemes were used in this study, one a subset of the other. In the first scheme, each user input action was assigned one code. The ten original codes were: error, drafting, split/join, perceptual, motor, query, drawing, system, backward one step, backward many steps. Some codes reflected actions taken on the part of the user to reduce different types of information processing loads such as perceptual (e.g., zooming reduces perceptual workload) and motor loads, while other codes ref[illegible]ed forward or backward movement to produce a design drawing, and finally, codes were applied to inputs which are an artifact of using a computer (system code).

When the codes were applied to the data it transpired that two of the codes were used very infrequently; for subsequent analysis, the codes of query and split/join were collapsed into the system code. For each user's encoded data file, we generated first-order transition matrices such as the one shown in figure 1. To apply the graphical summarization techniques, we further simplified the encoding techniques so that each input was classified into one of three states: a forward movement, a backward movement of magnitude n, or no movement. The inter-event interval analysis was applied to the time stamps of each input, and the encoded data was not involved. The data encoding for this study was performed manually and the analysis routines were written in FORTRAN.

3.2 Simulation and Rapid Prototyping System

We performed a second usability study which involved ESDA techniques on a beta version of a software system developed to support prototyping and simulations with graphical display. Five participants with varying degrees of computer and simulation experience participated. For this usability study we were able to collect only verbal protocols and a videotape of the display. The videotape was transcribed, including both participants' comments and actions, and then segmented and encoded with 25 predicates. The predicates were a mix of user interface object related encodings (e.g., menu selections, new interface, dialog completion, save); task activities related to building the simulation model, such as load model entity, object and process definition and manipulation, statement modification and selection, and process modification; and user activities (display examination, error, error correction and identification, materials reference, pauses, subject comments, requests for help, searches); and experimenter comment. Some of the predicates had values defined as well. The data was encoded using SHAPA. All the SDA techniques available in SHAPA were run on the encoded data, in addition to calculation of the more traditional usability measures such as frequency of usability tester intervention.

3.3 Airspace Scheduling System

Our most recent and sophisticated usability test, measured by attention given to developing an encoding language and assessing the applicability of the ESDA techniques, was performed on a prototype military airspace scheduling system (Cuomo and Bowen 1993). Four subject-matter experts and a user-system interface (USI) expert participated in this usability study. Verbal protocols, a videotape of the display, and time-stamped user keystrokes and mouse inputs were collected. The two data streams were integrated before application of an encoding scheme, allowing the user actions to be structured within a task context. Many of the users' psychological intentions were identified, allowing hierarchical segmentation of the input actions.

The 21 encodings used, which were loosely based on Norman's (1986) stages of user activity model, were divided into two levels and are shown in table 1. The first level included the semantic-level predicates; they provided information on the user's overall strategy, where and what types of errors were made and whether they were recovered from, what tasks were performed within each goal (task intentions), what computer steps (intentions to execute) were attempted in performing each task, and an evaluation of the success of each task and each execution sequence performed within that task. The articulatory-level encodings focused on the actual sequences of commands and user inputs for each intention to execute, and reflected generic user interface object usage. Each of the predicates also had detailed values which usually included the actual instance of the more generic predicate type. For instance, the "task intention" predicate included values for the goal number, the task number, and the name of the task the subject was performing (e.g., scheduling a particular mission, resolving a conflict). The "evaluate" predicate values included the goal, task, and execution number, the name of the task or execution being evaluated, and the evaluate state; possible states were abort, incomplete, OK, or wrong.

The articulatory-level values included the actual instance of the user interface object type being selected as well as the input device code (whether the mouse or keyboard was used, "m" or "k"). The "command" predicate, for example, had as values the actual command name selected followed by the input device code. The predicate "fields" had the name of the dialog box it resided in, the field name, and the user function performed in it (data entry into blank field, deleted information in the field, edited information in a populated field, or field was selected but not modified) as its three values. The encoding scheme and its hierarchical nature are illustrated in figure 3. SHAPA was again used as the encoding tool; its analysis routines were run on some of the encoded data files. We also experimented with the MRP tool by applying it to the participants' uncoded data files. The data files contained the numbered lines shown in the right-hand side of figure 3, but with the time information removed.

Table 1. Encodings Used for the Airspace Scheduling Study

Encoding	Definition
Goal	Scenario step.
Task intention (Int.task)	An intention to complete one task contributing to the completion of a goal.
Perception intention (Int.per)	An intention to improve the perceptibility of a display.
Intention to execute (Int.exe)	One computer step (may be comprised of multiple actions) leading to the completion of a task intention. Several steps may be required per task intention.
Evaluate (Eval)	The success with which the intention was accomplished.
Error in intention (Err.int)	The intention was incorrect and will not accomplish the goal.
Error in action specification (Err.acsp)	Wrong sequence of actions to accomplish the intention to execute.
Error in execution (Err. exec)	Manual, motor error in executing.
Error in perception (Err. per)	Break-down in human perceptual processing of information on a display.
Error in interpretation (Err. inter)	User fails to interpret system state correctly.
Error in evaluation (Err.eval)	User mistakenly thinks has or has not moved closer to the goal.
Recovered error (Rec.err)	Error was detected and recovered from.
Menu	A menu was opened
Command	A command was selected
List-select	An item is selected from a list
Button	A button was selected
Field	An action was taken in a field
Scroll	A scroll bar action was performed
Tape	A mission icon
Timebar	Manipulation of the timebar which controls horizontal scrolling in schedule
Form	The background area of the schedule

```
GOAL(1-setdate)                              Alright.  Okay, so I went to see that week."
    INT.TASK(1-1-setdate)                    001 11:32:39 000 Pressed Button on View Button
      INT.EXEC(1-1.1-setdate)                in Main Menu Bar
        MENU(view,m)                         002 11:32:41 002 Released Button on Date Button
        COMMAND(date,m)                      in View Menu
        BUTTON(date-cancel)                  003 11:32:43 002 Pressed Button on Cancel Button
      ERR.EVAL (e1*1-1.1-setdate-            in Date Dialog
      thought needed airspaces on display)   "Well, I probably need airspaces up there first."
      EVALUATE(1-1.1-setdate-abort)
                                             004 11:32:45 002 Pressed Button on  View Button
                                             in Main Menu Bar
                                             005 11:32:47 002 Released Button on  Change
      INT.EXEC(1-1.2-setlayout)              Layout Button in View Menu
                                             "Who am I again?  Phoenix"
        MENU(view,m)                         006 11:32:52 005 Pressed Button on  Undisplayed
                                             SUA List in General Layout Dialog
        COMMAND(layout,m)                    "Ah, Yankee 1."
          LIST_SELECT(layout-undis)          007 11:32:58 006 Pressed Button on  Add Button in
          BUTTON(layout-add)                 General Layout Dialog
          LIST_SELECT(layout-undis)          008 11:32:59 001 Pressed Button on  Undisplayed
          BUTTON(layout-add)                 SUA List in General Layout Dialog
                                             "Ah, Yankee 2."
        BUTTON(layout-ok)                    009 11:33:01 002 Pressed Button on  Add Button in
                                             General Layout Dialog
      EVALUATE(1-1.2-setlayout-ok)           010 11:33:02 001 Pressed Button on  OK Button in
                                             General Layout Dialog
```

Figure 3 Sample of the Encoded Data from the Airspace Usability Study

The first study using the CAAD system had a slightly different focus than the other two. It looked at the interaction between an architect's mental design activities and the use of a computer tool to support those activities. It was the only study that had variables: two different types of design tasks at three different levels of complexity. The second two studies were basic usability studies performed to identify areas in the system design that hindered users' task performance. There were no conditions to compare results across, and no pre-determined questions to answer or hypotheses to prove.

4 Applications of ESDA Techniques to the Collected Usability Test Data

Sequential data analysis techniques are potentially useful for analyzing usability data. To fully analyze the usability of a system, however, requires analysis of, and information on, the human-computer interaction process at several levels. At the highest level is information on the user's goals, intentions, and other high-level psychological processes. The next level involves the user's computer-use strategies and understanding how well the system meets the user's needs to carry out each task intention and convey information. Data files of actual input actions can provide only some of this information. Performing SDA on unencoded data files of user input actions will reveal information on usability at a low level, and only on certain types of problems. Repeated patterns, execution errors, etc. can be seen, but the context of the user's intentions are lost. Performing the wrong sequence of actions to accomplish an intention, for instance, is a type of error that will not be detected since the intention is not known.

Choosing a complex, hierarchical encoding scheme such as that used for the airspace scheduling study offered potential for data analysis at several levels. The generic encoding scheme used, however, was a poor match with SHAPA, which uses only predicate names for many of its routines and ignores the predicate's values. Much of the detailed information was in the predicate values, so the resulting outputs of the SDA techniques were ambiguous and hard to interpret. Furthermore, some information useful to assessing usability was not generated with any of the software supported techniques.

To fully analyze the airspace usability data, we were therefore forced to resort to manual methods (Cuomo et al. 1993). SHAPA also had file size limitations which forced us to break a single user's encoded data file into many separate files; hence report outputs had to be manually integrated. Below we discuss the different ESDA techniques we tried and explain the types of usability problems they were able to detect as well as their shortcomings.

4.1 Transition Matrices/Lag Sequential Analysis

The usefulness of the transition matrices varied from study to study. In the CAAD study, error-to-error transition probabilities turned out to be the most useful usability indicator as it reflected an important aspect of usability, error recovery. The most skilled partic[illegible]ant had a 0% probability of moving from one error state i[illegible] another, while some participants had probabilities as high as 21%. The simulation study usability analyst was able to discern only one usability problem from the transition matrix technique, and it was identified from the second order transition matrix output. For all users, the frequency of the combination "new interface, save, new interface" was high. This sequence reflected the modal nature of this system design, which only allowed users to save their work from one screen. Users would switch screens, perfo[illegible] their save function, and switch back to thei[illegible] original screen.

Transition matrices were also not very useful in analyzing the military airspace system data. One type of usability problem that could be extracted was the frequency with which a "menu" activity followed another "menu" activity, or any redundant double action. The menu example may indicate that men[illegible] being searched in an attempt to locate th[illegible]sired command. Repetitions of other actions could indicate a lack of system feedback or sl[illegible] system response time. For the airspace study, the generic nature of the predicate names provided no information on the actual instances of each action. From patterns such as "menu menu" or "button button" we could not determine which menus and buttons were activated, or even be sure if they were the same or different objects.

Analyzing data of this type is also difficult because of the large number of natural patterns that occur during the use of direct manipulation interfaces (e.g., command follows menu). The large number of these obvious or expected patterns, with their high frequencies, make it difficult to identify the often less-frequently occurring potential usability indicators; there is much noise in the data.

Reviewing the literature to dete[illegible]ine other's success using transition matrices in human-computer interaction analysis revealed that it was used most frequently to describe users' behavior patterns but not necessarily to determine usability problems. Good (1985) used uncoded command transition frequency data to determine the most common transitions between keys. This information was used in designing a new keyboard layout. Hammer and Rouse (1979) used the technique to assess how researchers used te[illegible]ors in writing their own programs and [illegible]s. They created 16 states for their Markov [illegible]el involving functions such as typing [illegible]itioning, deleting and inserting, and se[illegible]hing. They found differences in patterns of beh[illegible]or between editors, between tasks, and [illegible]nong users. They did not appear to use the data [illegible] identify usability problems. Finally, Penniman (1975) used the technique to analyze users' search behavior on an on-line retrieval system. He used both an 11 state and a 4 state model. Again, he found variations in users' patterns of behavior in comparing both sessions of different length and different parts of single sessions. As Penniman noted, the technique provides a quantitative, statistical rigor for comparing behavior across samples of different types, and thus helps to describe user be[illegible]ior.

For transition matrices to be useful in analyzing [illegible]ility data, behavior must be compar[illegible]ween two systems or across time, or there have to be certain defined, and detectable behavior patterns that alone can indicate usability problems. For instance, if users are studied for a long time, changes in their patterns of behavior can reflect system learnability or the effect of increased experience. Examples of behavior patterns that reflect potential usability problems were the error-to-error transition in the CAAD study, excessive switching between interfaces to perform a single function in the simulation study, or repeating the same action successively.

The lag sequential analysis revealed no usability problems for the airspace scheduling study. For the simulation study, which used more detailed predicates, this routine was found to be useful when run with respect to errors, as it helped to identify what activities preceded errors.

4.2 Frequency of Cycles

In addition to the predicate name-only limitation, SHAPA had the additional shortcoming of calculating frequency of cycles only between instances of the same predicate -- for example, goal to goal or menu to menu, but not menu to button. This greatly limited the usefulness of this analysis technique for our studies. In the airspace scheduling study, in one participant's (the USI expert) data we found a pattern that occurred 62 times: *intention to execute -> mission icon -> menu -> command -> evaluate -> intention to execute*. This was the basic sequence of activities needed to schedule the displayed airspace requests (also called mission icons). Acting on the intention, the user selects first a mission icon, then the schedule menu, and then the appropriate scheduling command (deny, approve, describe conflict, etc.). The user's cycle was completed with an evaluation of the success of the intention. How often this cycle recurs is important, as it indicates a highly repetitive pattern of behavior that could be reduced or eliminated by allowing a single command to be applied to many simultaneous objects. If the detailed values of each of these commands had been included, for instance the name of each mission icon, this cycle would not have occurred with high frequency, since the cycles would no longer be identical (unless the user scheduled the same mission 62 times, which is not likely but also cannot be determined from this analysis). On the other hand, if the detailed value information is not included, there could be important differences in these cycles which are not identified. We do not know, for example, if the command selected was the approve or deny command.

Another example illustrating the need for more analyst control over the level of cycle to be found was: *intention to execute -> menu -> command -> field -> field -> field -> field -> button -> evaluate -> intention to execute*. This cycle indicates that a dialog box was opened (menu, command), four data fields were accessed, and the box was closed (button, evaluate). We do not know specifically which dialog box was opened, which fields were accessed, or even whether they are the same or different fields. Little is therefore learned from this cycle. On the other hand, the generic cycle *intention to execute -> menu -> command -> button -> intention to execute*, if it occurs repeatedly, suggests that users are opening dialog boxes but not physically interacting with them or changing any data in their fields. This could mean that users are opening dialog boxes for the sole purpose of reading information contained in them, or that they opened the wrong box, realized it, and then closed it. The former could mean that some critical task information needs to be moved up to the main display or the next higher level, so it is more readily available. The latter may mean that the names of the commands for accessing the dialog boxes are confusing, so that users are having difficulty discriminating among them.

While for both instances we can detect a general trend, we do not know which dialog boxes are affected or how many. If we encoded the data along specific occurrences only, the general pattern would not show up. With the example given in the transition matrix section for the simulation usability study, the encoded command "save" allowed us to discover the problem of having to switch interfaces for the sole purpose of saving. If the users were also switching interfaces to perform some other command, we would miss this activity unless we selected that command to be encoded as well. Using the more generic predicate "command" as an encoding would have showed all occurrences of this pattern, but further investigation would be needed to determine which commands were involved. The optimal condition to maximize this routine's effectiveness would be to allow the analyst to run the frequency of cycles at a variety of levels.

4.3 Graphical Summarization Techniques

The task movement and task rate graphing techniques, which were used only in the CAAD study, were found to be a good way of summarizing the users' progress toward their goals, in terms of both the time and number of user inputs. Global usability problems, such as a high ratio of system commands to actual task commands, can be seen. These techniques, however, are too generic and high-level for directly indicating specific usability problems. Again, unless two or more systems are being compared, many users' data is needed to determine whether these problems are due to the system design or the users' use of the system.

We generated inter-event interval graphs for the CAAD and airspace studies. These graphs plot the time lags between each user input event. The presence of long delays may point the analyst to areas of human-computer interaction where the users are experiencing difficulty and which may warrant further investigation. This technique was useful in the CAAD study, as we could see the effect of task complexity on the frequency and duration of the long inter-event times. In this case, the long lag times were due to the designers using the time to problem solve and think up design solutions to meet the requirements. We were also able to divide the graph into discrete task activity areas, to see which activities were most affected by the increasing task complexity.

The technique was less effective for the airspace study, because this study was performed on a prototyped system and its software performance was not maximized. Thus, redrawing the complex screens caused a longer than normal time delay and introduced a lot of noise into our graphs. Some of the long time lags were, however, due to the users referencing written manuals and provided materials or attempting to recover from usability problems.

4.4 Value List and the Collection of Predicate Instances

The SHAPA value list routine generates a report on the number of occurrences of each constant for each predicate. One of two SHAPA routines where values were used, this is a detailed frequency counter that is always helpful for usability testing. The value list for the "task intention" predicate, for instance, lists all the instances of the users' task intentions and their frequencies. This was useful in the airspace study for counting error types, as six error classifications were used as predicates; it also provided counts of the specific instances of each error type within the six classifications. The value list is also helpful for providing information on the most frequently used commands, as well as the frequency of events which are considered to be usability problems. In the airspace study, for instance, when we encoded the data we tried to differentiate the reason for certain event's occurrence. Some events are executed routinely in the normal course of interaction, but sometimes the same events are performed, for example, to improve perceptibility. The distinction is important, because in one case it indicates a potential usability problem or an area that could be improved, while in the other case it may not. For instance, when the timebar was moved to control the part of the schedule that is viewed, the length of time for which the bar was manipulated was recorded, and the constant "p" was added as a value if users were thought to be performing the action to improve the perceptibility of missions on the display. Similarly, we had a predicate named "evaluate" in the airspace study with four states: OK, abort, incomplete and wrong. The frequencies of the latter three helped to point out when sequences of activity were not progressing well. By also including as a value the activity name that was being evaluated, we were able to correlate the evaluate state information to the activity being performed -- e.g., Evaluate (2-3-seeschedule-abort) [3].

Collection of predicate instances gathers segments that have been encoded with the same predicate. In the airspace study, we found this extremely useful for supporting our error analysis. We had defined six predicates related to errors. Using the collection of predicate instances for each error predicate for each subject, we could easily determine the number of errors of each type that occurred, along with the specific error descriptions and the line number where the predicate was located in the file. The line number was useful, as we often needed to go back to the original encoded file to collect more information on activities associated with the error.

Use of the error code across studies is also interesting. In the first two studies, the error code was used in the traditional way. If the system responded to a user input with an error message, the event is coded as a general error. In the airspace study, we used a more advanced error coding scheme. By integrating the data from the verbal protocols on the users' intentions and their actual inputs, we were able to not only assess errors of the physical or execution type, but also those in which the user's sequence of activities did not correspond with their intentions (errors in action specification), a type of error that does not cause the system to generate an error message. We also had classifications for other cognitive errors, such as errors in intention, and errors in perception, interpretation, and evaluation, as well as the more traditional execution error. This is a good example of where the encoding process is itself a form of analysis.

4.5 MRP Analysis

The MRP analysis technique was applied separately to the five participants' collected unencoded input data files from the airspace usability study. The data was very detailed, with each data line containing information about the user action (pressed, released, typed, moved), the object type (button, field, scroll bar, time bar), the specific name of the object (Bravo77, "Add" button, string typed), and the location of the object (in Build folder dialog box, in Create/Edit dialog, etc.). The five data files ranged from 1841 to 3317 lines in length, with 238, 386, 420, 422, and 534 MRPs generated. A sample MRP is shown in figure 4.

```
mrp#  6
 0)  Released Button on Bravo77 in an Sua Pane
 1)  Pressed Button on Bravo77 in an Sua Pane
 2)  Released Button on Bravo77 in an Sua Pane
 3)  Pressed Button on  Sua Description Field in Create/Edit Dialog
 4)  Pressed Button on  Sua Description Field in Create/Edit Dialog
 5)  Typed "0900" in Sua Description Field in Create/Edit Dialog
 6)  Pressed Button on  Sua Description Field in Create/Edit Dialog
 7)  Pressed Button on  Sua Description Field in Create/Edit Dialog
 8)  Typed "0900" in Sua Description Field in Create/Edit Dialog
 9)  Pressed Button on  Sua Description Field in Create/Edit Dialog
 10)  Pressed Button on  Sua Description Field in Create/Edit Dialog
 11)  Typed "0900" in Sua Description Field in Create/Edit Dialog
 12)  Pressed Button on  Sua Description Field in Create/Edit Dialog
 13)  Pressed Button on  Sua Description Field in Create/Edit Dialog
 14)  Typed "0900" in Sua Description Field in Create/Edit Dialog
 15)  Pressed Button on  Create Request Button in Create/Edit
Dialog
 16)  Pressed Button on  OK or Cancel Button in Confirmation Box
 17)  Pressed Button on Bravo77 in an Sua Pane
 18)  Released Button on Bravo77 in an Sua Pane
at:   1661   1679   1755
Total number of positions =   3.
```

Figure 4 Sample Output from the MRP Tool Showing a Single MRP of Length 19, Occurring in 3 Different Positions.

To assess the MRPs, we tried using the heuristics provided by Siochi et al. (1991) to narrow down the number of MRPs that need to be examined. These included examining the longest MRPs, the most frequently occurring, and anomalies departing from the expected patterns of MRPs (expected patterns are few long MRPs and many short MRPs). Unfortunately, this limited set of MRPs did not reveal any usability problems, and we had to examine every generated MRP. In general, the more meaningful patterns seemed to relate to five types of activity: scroll bar movement, time bar movement, data editing sequences in dialog boxes, list selection, selecting or moving the mission icons, and approving mission sequences.

For some participants' data sets it was harder to find potential usability problems in the generated MRPs, because the participants did not work methodically. Few meaningful repetitious patterns could be identified among the many repeating sequences identified. Some repetitious patterns concerning usability issues could be seen, however, in the MRPs relating to the ability to select only individual items from a list, having to schedule each mission part and each mission individually, and the dialog box problem. The dialog box problem was previously discussed as the case in which SHAPA generated a high-level cycle showing dialog boxes being opened and then immediately closed, but yielding no information on which dialog box was used. With MRP analysis, some MRPs were generated showing the actual patterns of actions for this occurrence for the create/edit dialog box. To find all the specific occurrences, however, involves looking across all the MRPs, because problems of the same type, or even identical patterns, are not necessarily grouped together. If the sequence of interest was sometimes part of a larger repeating sequence, that larger sequence would be located in a different MRP. Given the large number of MRPs generated, it can be difficult to find all instances.

This technique provides only one potential indicator of usability problems -- that of repetitive sequences of activities. Problems with the technique include the random approach to pattern identification, which precludes frequency counts of a particular pattern, and the patterns identified, which are totally context free and unrelated to any task or user interface sequences. Many usability problems can only be identified if user intentions are known, and this technique will not find those. It also generates a large amount of output with a lot of noise; e.g., many MRPs were generated relating to scroll bar activity or tabbing through data fields.

The technique does have some good points. Many of the problem specifics missed by SHAPA's frequency of cycles because values were not considered were made somewhat apparent with this analytic technique (particularly since we knew what to look for), since it was operating on much more detailed data. The technique is relatively easy and quick to apply if the appropriate data can be collected; no data encoding is required. The program had no trouble accommodating large data files. Finally, the command usage statistics could be useful for providing frequency information at a very detailed level; the formatting of this particular output, however, could use some improvement.

5 Conclusions

We hoped to shed light on the types of system usability information each of the sequential data analysis techniques revealed, the trade-off of questions answered and level of encoding used, and whether it was worth applying the techniques. Overall, we conclude that we did not have a great deal of success in effectively utilizing most of the sequential data analysis techniques for analyzing our usability study data. Many interacting variables affect what can be learned from application of the techniques, including the types of data that can be collected, the encoding scheme used if the data is encoded, the flexibility with which the SDA routines can be applied, and the types of usability problems to be addressed. If we had to rank-order the techniques discussed here from best to worst for identifying usability problems based on our experiences, we would put hierarchical data encoding as the most useful activity, and MRP analysis second (because it is quick and easy to apply), followed the value list, collection of predicate instances, frequency of cycles, transition matrices, and lag sequential analysis. To indicate overall system usability, the graphical techniques are somewhat useful.

Nevertheless, we are still attracted to the idea of pattern analysis and analytic techniques for analyzing usability data and feel the problems we encountered are due mostly to limitations of the currently available software packages -- specifically, their lack of flexibility in specifying the data parts and levels for the routines to act on. To effectively utilize routines such as transition matrices, lag sequential analysis, and frequency of cycle analysis, the usability analyst needs to be able to apply the routines at various levels, without having to recode the data. As we have shown, we need to be able to identify both generic and specific patterns in the data with a single tool. This could be achieved by having frequency of cycles, lag sequential, and transition matrices routines operate on both the predicates and their values, permitting use of wild cards for particular values. This would provide the flexibility needed to get at a large variety of useful patterns, or to follow up leads indicated by the general patterns.

For the frequency of cycles routine, allowing identification of both a start and an end predicate would allow analysts to better define the types of patterns we want the system to find. There seem to be at least two types. One is a task activity pattern in which we might want to specify task-related start and stop points, such as between specific user intentions to execute or task intentions and their corresponding evaluate state. This would depict activity within a task-domain cycle. It would also be helpful to be able to identify user-interface object usage patterns across task activities. Here we would like to specify a cycle, such as from dialog box opening to closing, which would find all dialog box usage patterns along with the usage of objects contained within them.

The problem with the MRP routines which work on the unencoded command files is the loss of context or user intention information. The data cannot be easily aggregated along task lines, and the users goals and intentions are not known. This makes identification of many types of usability problems very difficult. Siochi et al. (1991) had to supplement their MRP analysis of the GIPSY system by interviewing the users.

When using verbal protocols in conjunction with data logging techniques, the user's thought processes can be extracted to supplement the logged mouse/keystroke data during encoding; this puts structure on what would have been otherwise difficult to interpret data. The process of encoding the data was found to be the most useful analytic activity, particularly in the airspace study, where we used codes that allowed us to hierarchically break down the user input sequence into goals, tasks, intentions to execute, actual sequences of inputs within each task and execute intention, and evaluation of each activity. We also learned much from the detailed error codes used. This coding scheme did not lend itself to use of SHAPA's SDA techniques, but neither did the coding schemes we tried for the other studies. Moreover, with the encoded data in this easy-to-read form, patterns were easily detectable by the usability analyst. In fact, it was easier first to manually detect patterns, then figure out which SDA analysis routine to run and with what parameters, and finally run the SDA routines to generate hard frequency counts. To be able to say a repetitive pattern occurred 62 times in 90 minutes creates much more impact than just noting that such a pattern exists.

During the airspace study we also identified and manually extracted other measures of interest that we felt reflected system usability but were not supported by any software packages, such as the number of computer actions per intention to execute. Some patterns of user activity could be recognized by human analysts might not be identified by a software pattern recognizer because they do not repeat exactly or regularly. For instance, users often looked up information on a mission icon in a dialog box before scheduling it, but did not always do so sequentially or with the same exact set of actions; also, the mission icon was different in every case. The computer programs do not identify these as repetitive activities.

Software to support application of SDA techniques for usability testing is still in its infancy. As programs become more flexible and powerful, and usability analysts identify measures and routines of interest and use to them, these tools should become more effective.

Acknowledgments

The simulation and prototyping usability study was conducted by Janet S. Blackwell. Support for the airspace study was provided by Charles D. Bowen, Scott E. Blomquist, and Elizabeth Wadick.

References

Cuomo, D. L. and Bowen, C. D. 1993, Measures of User-System Interface Effectiveness: An Encoding Scheme and Indicators for Assessing the Usability of Graphical, Direct Manipulation Style Interfaces, MITRE Technical Report 92B0000047, Vol. 3, Bedford, MA.

Cuomo, D. L. and Sharit, J. 1989, A Study of Human Performance in Computer-Aided Architectural Design, *International Journal of Human-Computer Interaction*, 1, 69-107.

Good, M. 1985, The Use of Logging Data in the Design of a New Text Editor, in *Proceedings of CHI '85, Conference on Human Factors in Computing Systems*, Apr. 14-18, San Francisco (ACM, New York), 93-97.

Hammer, J. M. and Rouse, W. B. 1979, Analysis and Modeling of Freedom Text Editing Behavior, in *Proceedings of the International Conference on Cybernetics and Society*, Denver, CO, 659-664.

Holleran, P. A. 1991, A Methodological Note on the Pitfalls in Usability Testing, *Behaviour & Information Technology*, 10, 345-357.

James, J. M., Sanderson, P. M. and Seidler, K. S. 1990, SHAPA Version 2.0 Instruction Manual and Reference, EPRL-90-16/M, University of Illinois at Urbana-Champaign.

Kemeny, J. G. and Snell, J. L. 1960, *Fi ite Markov Chains* (New York: Van Nostrand Co.).

Newell, A. and Simon, H. A. 1977, *Human Problem Solving* (New Jersey: Prentice-Hall, Inc.).

Norman, D. A. 1986, Cognitive Engineering, in D. A. Norman and S. W. Draper (eds) *User Centered System Design: New Perspectives on Human-Computer Interaction.*, (Hillsdale, NJ: Lawrence Erlbaum Associates).

Penniman, W. D. 1975, A Stochastic Process Analysis of On-Line User Behavior, in *Proceedings of the 38th Annual ASIS Meeting*, Boston, 147-148.

Sanderson, P. M. 1991, ESDA: Exploratory Sequential Data Analysis, EPRL-91-04, University of Illinois at Urbana-Champaign.

Sanderson, P. M., James, J. M., and Seidler, K. S. 1989, SHAPA: An Interactive Software Environment for Protocol Analysis, *Ergonomics*, 32, 1271-1302.

Sanderson, P. M., Watanabe, L. M., James, J. M. 1991, Visualization and Analysis of Complex Sequential Data Using SHAPA (MAC), Proceedings of the 3rd European Conference on Cognitive Science Approaches to Process Control, September, 121-135.

Siochi, A. C. and Ehrich, R. 1991, Computer Analysis of User Interfaces Based on Repetition in Transcripts of User Sessions, *ACM Transactions on Information Systems*, 9, 309-335.